

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

**Event Consumers for an Event Management System**

Inventors:

Ashvinkumar J. Sanghvi

Howard M. Hance

Lev Novik

Patrick R. Kenny

Michael A. Thatcher

Alexander E. Nosov

1 **RELATED APPLICATIONS**

2 This application claims the benefit of U.S. Provisional Application No.  
3 60/210,330, filed June 7, 2000.

4  
5 **TECHNICAL FIELD**

6 The present invention relates to computing systems and, more particularly,  
7 to event consumers, such as application programs, that receive events generated by  
8 components, services and applications in a computing environment.

9  
10 **BACKGROUND**

11 Computer systems, such as servers and desktop personal computers, are  
12 expected to operate without constant monitoring. These computer systems  
13 typically perform various tasks without the user's knowledge. When performing  
14 these tasks, the computer system often encounters events that require a particular  
15 action (such as logging the event, generating an alert for a particular system or  
16 application, or performing an action in response to the event). Various  
17 mechanisms are available to handle these events.

18 A computing enterprise typically includes one or more networks, services,  
19 and systems that exchange data and other information with one another. The  
20 enterprise may include one or more security mechanisms to safeguard data and  
21 authenticate users and may utilize one or more different data transmission  
22 protocols. At any particular time, one or more networks, services or systems may  
23 be down (e.g., powered down or disconnected from one or more networks).  
24 Networks, services or systems can be down for scheduled maintenance, upgrades,  
25

1 overload or failure. Application programs attempting to obtain event data must  
2 contend with the various networks, services, and systems in the enterprise when  
3 they are down. Additionally, application programs must contend with the security  
4 and network topology limitations of the enterprise as well as the various protocols  
5 used in the enterprise.

6 A typical computing environment includes multiple event consumers (i.e.,  
7 applications and other routines that use various event data generated by one or  
8 more event sources or event providers). These event consumers are typically  
9 implemented by network administrators or other individuals responsible for the  
10 operation of the computing environment. An administrator generally implements  
11 many different event consumers to properly handle the various events generated  
12 throughout the computing environment. A particular administrator may be  
13 responsible for a portion of the computing environment, such as the computing  
14 devices in a particular building or the computing devices associated with a  
15 particular department in an organization. Different administrators may implement  
16 duplicate (or substantially similar) event consumers stored in different portions of  
17 the computing environment. The creation of duplicate event consumers is  
18 wasteful of the administrators' time and wastes storage space by storing redundant  
19 event consumers.

20 The system and method described herein addresses these limitations by  
21 providing a standard set of event consumers for handling various common events  
22 (i.e., events that are likely to be handled by multiple administrators). The system  
23 and method described herein also provides a standard schema that allows event  
24  
25

1 consumers to use event data without requiring knowledge of the source of the  
2 event data.

### 3 4 5 **SUMMARY**

6 The system and method described herein provide a standard set of  
7 commonly used event consumers, thereby eliminating the need for an  
8 administrator to implement those event consumers. Providing a standard set of  
9 common event consumers also reduces the number of redundant consumers stored  
10 throughout the computing environment. The use of a standard schema for  
11 defining event data allows an event consumer to accept and used event data from  
12 any event source. The event consumer does not require any knowledge about the  
13 event source to process the event. Similarly, the event source does not require any  
14 knowledge of the event consumer to generate event data.

15 In one embodiment, an event management system includes an email  
16 consumer for handling email, a paging consumer, an active scripting consumer, a  
17 log file consumer, an event log consumer, and a command line consumer.

18 In a described embodiment, a procedure includes creating an instance of an  
19 event filter which filters events based on event filter properties. The procedure  
20 also includes creating an instance of an event consumer which defines an action  
21 and creating a binding between the instance of the event filter and the instance of  
22 the event consumer.

23 In a particular embodiment, a schema includes at least one event consumer  
24 class that represents a consumer of an event. The schema also includes at least  
25 one event filter class that represents event filtering parameters and at least one

1 binding class that represents the association of at least one event consumer and at  
2 least one event filter.

### 3 4 5 **BRIEF DESCRIPTION OF THE DRAWINGS**

6 Fig. 1 illustrates a block diagram of a system that receives event  
7 information from multiple event providers and provides event information to  
8 multiple event consumers.

9 Fig. 2 illustrates a block diagram of a system that receives events and logs  
10 those events to an event log.

11 Fig. 3 is a flow diagram illustrating a procedure for implementing event  
12 consumers.

13 Fig. 4 illustrates the binding of an instance of an event consumer with an  
14 instance of an event filter.

15 Fig. 5 illustrates the forwarding of events from multiple event sources to a  
16 common event target.

17 Fig. 6 illustrates an example of a suitable operating environment in which  
18 the event-handling system and method may be implemented.

### 19 20 21 **DETAILED DESCRIPTION**

22 The system and method described herein provide a standard set of event  
23 consumers for use throughout a computing environment. This standard set of  
24 event consumers includes an email consumer (such as an SMTP consumer), a  
25 script consumer, a paging consumer, a log to file consumer, an event log

1 consumer, a command line consumer, and an event forwarding consumer. The use  
2 of these standard event consumers eliminates the need for administrators to  
3 implement such consumers and reduces the number of redundant (or substantially  
4 similar) consumers in the computing environment. A standard schema is  
5 described for defining and handling event data. This standard schema allows an  
6 event consumer to utilize event data from any source without requiring any  
7 knowledge of the event source. Additionally, the event data can be generated  
8 without knowledge of the event consumer that may utilize the event data.

9 Web-Based Enterprise Management (WBEM) provides uniform access to  
10 management information throughout an enterprise. WBEM is an industry  
11 initiative to develop technology for accessing management information in an  
12 enterprise environment. This management information includes, for example,  
13 information on the state of system memory, inventories of currently installed client  
14 applications, and other information related to the status of the system. A particular  
15 embodiment of the event-handling system is implemented using Windows<sup>®</sup>  
16 Management Instrumentation (WMI) developed by Microsoft Corporation of  
17 Redmond, Washington, which provides an infrastructure to handle various events  
18 generated by event sources throughout an enterprise.

19 The WMI technology enables systems, applications, networks, and other  
20 managed components to be represented using the Common Information Model  
21 (CIM) designed by the Distributed Management Task Force (DMTF). CIM is an  
22 extensible data model for representing objects that exist in typical management  
23 environments. CIM is able to model anything in the managed environment,  
24 regardless of the location of the data source. The Managed Object Format (MOF)  
25

1 language is used to define and store modeled data. In addition to data modeling,  
2 WMI provides a set of base services that include query-based information retrieval  
3 and event notification. Access to these services and to the management data is  
4 provided through a single Component Object Model (COM) programming  
5 interface.

6 The WMI schema includes multiple classes. Each WMI class is associated  
7 with a system or subsystem in, for example, an enterprise. WMI classes define the  
8 basic units of management. Each WMI class is a template for a type of managed  
9 object. For example, Win32\_DiskDrive is a model representing a physical disk  
10 drive. For each physical disk drive that exists, there is an instance of the  
11 Win32\_DiskDrive class. WMI classes may contain properties, which describe the  
12 data of the class and the methods (which describe the behavior of the class).

13 WMI classes describe managed objects that are independent of a particular  
14 implementation or technology. WMI includes an eventing subsystem that follows  
15 the publish-subscribe model, in which an event consumer subscribes for a  
16 selection of events (generated by one or more event providers) and performs an  
17 action as a result of receiving the event. WMI also provides a centralized  
18 mechanism for collecting and storing event data. This stored event data is  
19 accessible by other systems via WMI tools and/or application programming  
20 interfaces (APIs).

21 Although particular embodiments are discussed herein as using WMI,  
22 alternate embodiments may utilize any enterprise management system or  
23 application, whether web-based or otherwise. The event providers and event  
24 consumers discussed herein are selected for purposes of explanation. The  
25

1 teachings of the present invention can be used with any type of event provider and  
2 any type of event consumer. Additionally, the event-handling system and method  
3 described herein can be applied to any type of enterprise or other arrangement of  
4 computing devices, applications, and/or networks.

5 Fig. 1 illustrates a block diagram of a system 100 that receives event  
6 information from multiple event providers 108 (i.e., event sources) and provides  
7 event information to multiple event consumers 102 (i.e., the users of the event  
8 data). System 100 includes a WMI module 106, which receives event data from  
9 multiple event sources 108 and receives requests for information (e.g., notification  
10 of particular events) from multiple event consumers 102. Event sources 108 may  
11 include, for example, managed nodes or managed systems in a network. The  
12 multiple event sources are identified as event providers 110. The multiple event  
13 consumers are identified as applications 104.

14 WMI module 106 shown in Fig. 1 represents the managed node layer of the  
15 WMI module. As discussed below, the WMI module 106 may also include a  
16 central store layer, which may include user interface functionality. The different  
17 layers of WMI module 106 manage different types of activities and/or perform  
18 different types of functions.

19 Event providers 110 include, for example, systems, services or applications  
20 that generate event data. An exemplary event provider is a disk drive (or an  
21 application that monitors the status of a disk drive). The disk drive may generate  
22 an event indicating the available storage capacity on the disk drive or indicating  
23 the amount of data currently stored on the disk drive. The disk drive may also  
24 generate an event indicating that the disk drive is nearly full of data (e.g., when  
25



1 ninety-five percent or more of the disk drive's capacity is used).

2 Event consumers 102 may request to be notified of certain events (also  
3 referred to as "subscribing" to an event). An example event consumer is an  
4 application that manages multiple storage devices in an enterprise. The  
5 application may request to receive events generated by any of the disk drives or  
6 other storage devices in the enterprise. The application can use this event  
7 information to distribute storage tasks among the multiple storage devices based  
8 on the available capacity of each device and/or the quantity of read or write  
9 requests received by each storage device.

10 Fig. 2 illustrates a block diagram of a system 150 that receives events and  
11 logs those events to an event log. System 150 includes a central store layer of  
12 WMI module 106, which is coupled to multiple user interface (UI) applications  
13 152. UI applications 152 are used to access WMI module 106 to retrieve data,  
14 manage systems, and configure various enterprise management parameters. The  
15 central store layer of WMI module 106 provides for the centralized logging and  
16 storage of event data received from various nodes and various networks in an  
17 enterprise. WMI module 106 is also coupled to receive events 162 from one or  
18 more event sources. For example, events may be received from the managed node  
19 layer of WMI module 106, discussed above with respect to Fig. 1, from an event  
20 forwarding application (e.g., application 104), or from one or more event  
21 providers (e.g., event provider 110).

22 System 150 also includes a set of policies 160, which are accessible by  
23 WMI module 106. Policies 160 may control the configuration of one or more  
24 systems in the enterprise. Other policies may define various activities, such as  
25

1 event filtering, event correlation, and the forwarding of events to particular  
2 devices or applications. A database 156 is coupled to WMI module 106.  
3 Database 156 stores various information related to the enterprise. For example,  
4 database 156 can store event data (i.e., creating an event log), policy data, and  
5 enterprise configuration information.

6 WMI module 106 is also coupled to an event log 158. The event log 158  
7 uses WMI features to provide a distributed architecture that is capable of selecting,  
8 filtering, correlating, forwarding, storing, and delivering event data in an  
9 enterprise. The event log 158 allows users, such as administrators, to request data  
10 related to a particular event, request data from a particular node or device in the  
11 enterprise, define the manner in which events are correlated with one another,  
12 define how certain events should be forwarded, and define how to store event data.  
13 Data requests may be accessed from the event log 158 using, for example, a  
14 particular UI application 152. The event log 158 uses an event provider model  
15 that allows an application, device or driver to generate events.

16 The event log 158 provides a policy-based administration of the enterprise.  
17 The policy infrastructure allows administrators to set a policy in the Directory  
18 Service (DS) and the WMI module ensures that the proper set of WMI objects  
19 (e.g., filters, bindings, correlators, consumers, and configuration objects) are  
20 delivered to the proper devices or applications in the enterprise.

21 Table 1 below identifies various types of event providers available in a  
22 particular embodiment. Additionally, the table includes a description of the events  
23 generated by each event provider. For example, the Win32 Provider generates  
24 events that include information related to the operating system, computer system,  
25

peripheral devices, file systems, and security for a particular device (such as a computer system) in the enterprise.

TABLE 1

Event Provider	Description of Events Provided
Win32 Provider	Supplies information about the operating system, computer system, peripheral devices, file systems, and security.
WDM Provider	Supplies low-level Windows Driver Model (WDM) information for user input devices, storage devices, network interfaces, and communications ports.
Event Log Provider	Allows the reading of Windows NT event log entries, controls the configuration of event log administrative options, and event log backup.
Registry Provider	Allows registry keys to be created, read, and written. WMI events can be generated when specified Registry keys are modified.
Performance Counter Provider	Exposes the raw performance counter information used to compute various performance values.
Active Directory Provider	Acts as a gateway to information stored in Microsoft Active Directory services. Allows information from both WMI and Active Directory to be accessed using a single API.
Windows Installer Provider	Supplies information about applications installed with the Windows Installer.
SNMP Provider	Acts as a gateway to systems and devices that use SNMP for management. Allows SNMP traps to be automatically mapped to WMI events.

Fig. 3 is a flow diagram illustrating a procedure 200 for implementing event consumers. Initially, a customer (such as an administrator) creates an instance of an event filter (block 202) to specify the events that should trigger an action. Next, the customer creates an instance of an event consumer for the desired action (block 204) to specify the parameters of the action to be taken. The customer then creates an association that binds the event filter with the event consumer (block 206). This association indicates that the specified action is to be performed when events matching the event filter criteria occur. A customer can bind one event filter to several event consumers, indicating that when matching events occur, several actions are to be performed. Similarly, a customer can bind one event consumer to several event filters such that the specified action is performed whenever events matching any of the event filters occur.

Referring again to Fig. 3, if additional events are to be associated with the event class (block 208), then procedure 200 branches to block 210, where the customer creates an association that binds one or more additional event filters with the event consumer. If additional event consumers are to be associated with the event filter (block 212), then procedure 200 branches to block 214, where the customer creates an association that binds the event filter with one or more additional event consumers. The procedure illustrated in Fig. 3 may be repeated, as necessary, to implement additional event consumers.

Fig. 4 illustrates the binding of an instance of an event consumer with an instance of an event filter. An event consumer 300 is a class that includes one or more properties, labeled Property 1, Property 2, ... Property N. The event consumer properties specify one or more actions to be performed in response to

1 the occurrence of a particular event. Event consumer 300 includes multiple  
2 subclasses 304. In a particular embodiment, the subclasses of event consumer 300  
3 are an SMTP consumer for handling mail messages, a script consumer for  
4 executing scripts, a paging consumer for initiating paging messages, a log to file  
5 consumer to log event data to a file, an NT event log consumer to log event data to  
6 an NT event log, a command line consumer for executing command line  
7 instructions, and an event forwarding consumer to forward events from one system  
8 to another. These various subclasses are discussed in greater detail below.

9 An event filter 306 is a class that includes one or more properties, labeled  
10 Property 1, Property 2, ... Property N. The event filter properties identify the  
11 events that should cause one or more actions to be performed. A binding 308 is a  
12 class that includes at least two properties, labeled Property 1 and Property 2. An  
13 instance of binding 308 creates an association between an instance of an event  
14 consumer 302 and an instance of an event filter 306. The two binding properties  
15 identify event consumer 302 and event filter 306 as the two instances that are  
16 bound together by binding 308. After event consumer 302 and event filter 306 are  
17 bound together, the event consumer is executed when the filter event occurs.  
18 Specific examples of event consumers, event filters and the bindings between  
19 them are discussed below.

20 If the action designated by a consumer fails to execute (the definition of a  
21 failure is specified with every consumer type), WMI will generate a  
22 ConsumerFailureEvent event. The event contains as properties both the original  
23 event that failed to be delivered, and the logical consumer instance representing  
24 the failing consumer. Interested clients can register to receive these events, or  
25 perform specific actions upon their receipt.

1 In a number of event consumers, an occasion arises to create a string that is  
2 partly configured in the event consumer instance, and partly derived from the  
3 event in question. For these cases, a template language similar to the NT  
4 environment variable specification is used. Following are some examples of the  
5 syntax used in the templates:

6  
7 “Some Text Here” will always produce “Some Text Here”

8 “%CPUUtilization%” will always produce the value of the CPUUtilization  
9 property of the event being delivered, converted to a string if necessary, e.g. “90”

10 “The CPU utilization of my processor is %CPUUtilization% at this time” will  
11 embed the value of the CPUUtilization property of the event into the string,  
12 producing something like “The CPU utilization of my processor is 90 at this time”.

13 “%TargetInstance.CPUUtilization%” will retrieve the CPUUtilization property  
14 of the embedded instance in TargetInstance.

15 “%%” produces a single % sign

16 If the property being retrieved is an array, the entire array will be produced, in  
17 the format of (1, 5, 10, 1024). If there is only one element in the array, parenthesis  
18 will be omitted. If there are no elements in the array, “()” will be produced.

19 If a property is an embedded object, the MOF representation of the object will  
20 be produced (similar to GetObjectText).

21 If a property of an array of embedded objects is requested, it is treated as a  
22 property with the value of an array. For instance,  
23 %MyEvents.TargetInstance.DriveLetter% could produce ‘(“c:”, “d:”)’ if  
24 MyEvents is an array of embedded instance modification events.  
25

1 If a property of a consumer class is interpreted to be a template according to  
2 the above rules, it is marked with a [template] qualifier.

3 In a particular embodiment, the event consumers described herein are  
4 implemented as dynamically linked libraries (DLLs), except for the active  
5 scripting consumer, which is discussed below. In this embodiment, the event  
6 consumers execute in the security context of the LocalSystem. Further, only  
7 authorized users (e.g., administrators) are permitted to configure standard event  
8 consumers of the type described herein. The list of authorized users may vary  
9 from one event consumer to another.

10 Details regarding the various event consumers and their associated  
11 properties are described below.

### 12 13 Log File Event Consumer

14 This event consumer will write customized strings to a text log file  
15 whenever events are delivered to the consumer. The strings will be separated by  
16 end-of-line sequences. The logical consumer class for the log file event consumer  
17 is:

```
18  
19 class LogFileEventConsumer : __EventConsumer  
20 {  
21     [key] string Name;  
22     string Filename;  
23     [template] string Text;  
24     uint64 MaximumFileSize;  
25     boolean IsUnicode;  
26 };  
27
```

28 where

29 “Filename” is the name of the file to which the log entries are appended

1 “Text” is the template (as described above) for the text of the log entry

2 “MaximumFileSize” is the maximum size (in bytes) that the log file will be  
3 allowed to grow. If the primary file exceeds its maximum size, its contents will be  
4 moved to another file, and the primary file will be emptied. Default is 0, which  
5 will be interpreted as no limit.

6 “IsUnicode” is true if the file in question should be a UNICODE (as opposed  
7 to MBC) file.

8  
9 The naming structure for the backup files will be as follows:

10  
11 If the original filename is 8.3, the extension will be replaced by a string of the  
12 format “001”, “002”, etc, with the smallest number larger than all those used  
13 chosen each time (unless “999” is used, in which case the smallest unused number  
14 is chosen).

15 If the original filename is not 8.3, the suffix described above will be appended  
16 to the filename.

17  
18 The file is opened for shared write access. Any failure to open or write to  
19 the file will be considered a failure of the action (this includes the case where  
20 another application has the file opened with exclusive access). The user who  
21 created the binding as identified by the CreatorSID property must have write  
22 access to the file in question at the time the event is generated in order for the  
23 consumer to write to the file.



## Command-Line Event Consumer

This event consumer can launch an arbitrary process whenever an event is delivered to the consumer. The process will be launched in the LocalSystem security context. The logical consumer class for the command-line event consumer is:

```
class WMI_CommandLineEventConsumer : __EventConsumer
{
    [key] string Name;
    [not_null] string ExecutablePath;
    [template] string CommandLineTemplate;
    boolean UseDefaultErrorMode = FALSE;
    boolean CreateNewConsole = FALSE;
    boolean CreateNewProcessGroup = FALSE;
    boolean CreateSeparateWowVdm = FALSE;
    boolean CreateSharedWowVdm = FALSE;
    sint32 Priority = 32;
    string WorkingDirectory;
    string DesktopName;
    string WindowTitle;
    uint32 XCoordinate;
    uint32 YCoordinate;
    uint32 XSize;
    uint32 YSize;
    uint32 XNumCharacters;
    uint32 YNumCharacters;
    uint32 FillAttribute;
    uint32 ShowWindowCommand;
    boolean ForceOnFeedback = FALSE;
    boolean ForceOffFeedback = FALSE;
    boolean RunInteractively = FALSE;
    uint32 KillTimeout = 0;
};
```

where all the parameters are as documented in the Win32 Software Developers Kit (SDK), available from Microsoft Corporation of Redmond, Washington, for CreateProcess function (and its parameter STARTUPINFO), except:

1 “CommandLineTemplate” is a template (as described above), e.g.  
2 “C:\winnt\runreport %TargetInstance.DriveLetter%”

3 “RunInteractively” can be set to TRUE to force the process to be launched in  
4 the interactive winstation. Otherwise, the process is launched in the default  
5 service winstation. This property overrides the “DesktopName”, which can also  
6 be used to select a specific winstation and desktop.

7 “KillTimeout” can be specified to have WinMgmt kill the launched process  
8 after a specified number of seconds.

9  
10 Failure to launch the process (CreateProcess) will be considered a failure of  
11 the action. Failure return code from the process will not be considered a failure of  
12 the action. In one embodiment, only local administrators are allowed to register  
13 this event consumer, because the process in question will run as LocalSystem.

#### 14 15 NT Event Log Event Consumer

16 NT event log event consumer will log a specific message to the NT Event  
17 Log whenever an event is delivered to the consumer.

18 The NT Event Log requires that the message text of all entries be placed in  
19 a message DLL, properly installed on the system on which events are logged.  
20 This event consumer does not change this requirement. It is still the responsibility  
21 of the customer to properly register an NT Event Log “Source” with the message  
22 texts; once that is done, however, this consumer can log NT Event Log entries  
23 based on that source whenever designated WMI events occur.  
24  
25

1 The logical consumer class for the NT event log event consumer is:

```
2 class NTEventLogEventConsumer : __EventConsumer
3 {
4     [key] string Name;
5     string UNCServerName;
6     string SourceName;
7     [not_null] uint32 EventID;
8     uint32 EventType = 1;
9     uint32 Category;
10    [template] string InsertionStringTemplates[] = {""};
11 };
12
```

13 where

14 “UNCServerName” is the name of the machine on which to log the event, or  
15 NULL if the machine is a local server.

16 “SourceName” is the name of the NT Event Log Source in which the message  
17 is to be found. As mentioned above, the customer is assumed to have registered a  
18 DLL with the necessary messages under this source.

19 “EventID” is the id of the event message in the Source.

20 “EventType” is the type of the event being generated, e.g. Informational,  
21 Warning, or Error.

22 “Category” is as documented in the Win32 SDK ReportEvent function.

23 “InsertionStringTemplates” is an array of templates whose values are used as  
24 the insertion strings for the event log record.

25 Failure to write the event (ReportEvent) is considered a failure. Lack of  
installed message DLL for the Source, use of out-of-range IDs, or invalid number  
of Insertion Strings are not considered failures.

## Active Scripting Event Consumer

The active scripting event consumer will execute a predefined script in an arbitrary scripting language whenever an event is delivered to the consumer. While the text of the script itself is specified in the event consumer instance, the script will have access to the event instance in the script environment variable TargetEvent. For instance,

MsgBox TargetEvent.TargetInstance.DriveLetter

in VBScript would bring up a message box with the drive letter of the event in the message box.

The scripts will execute in the security context of LocalSystem. In a particular embodiment, as a security measure to prevent abuse, only a local system administrator or a domain administrator may configure the active scripting event consumer. The access rights are not checked until runtime. Once the consumer is configured, any user may trigger the event that causes the script to be executed.

The logical consumer class for the active scripting event consumer is:

```
class ActiveScriptEventConsumer : __EventConsumer
{
    [key] string Name;
    string ScriptingEngine;
    string ScriptText;
    string ScriptFileName;
    [units("seconds")] uint32 KillTimeout;
};
```

where

1 “ScriptingEngine” is the ProgID of the scripting engine to use, e.g. “VBScript”  
2 or “JScript”.

3 “ScriptText” is the text of the script to execute. “ScriptText” may be NULL, in  
4 which case ScriptFileName is used.

5 “ScriptFileName” is the name of the file from which the text of the script is  
6 read, unless ScriptText is specified. Only one of “ScriptText” or  
7 “ScriptFileName” may have a value.

8 “KillTimeout” specifies the number of seconds after which the script will be  
9 terminated if not already finished. Killing a script via the timeout is considered an  
10 error. If “KillTimeout” is zero or NULL, the script will not be terminated.

11  
12 There is also a global configuration class in the root\cimv2 namespace that  
13 applies to all instances of the consumer:

14  
15 class ScriptingStandardConsumerSetting : CIM\_Setting  
16 {  
17 string SettingID = "ScriptingStandardConsumerSetting";  
18 string Caption = "Scripting Standard Consumer Setting";  
19 string Description = "Registration data common to all instances of the  
20 Scripting Standard Consumer";  
21 uint32 MaximumScripts;  
22 uint32 Timeout;  
23 };  
24

25  
where

“SettingID”, “Caption”, and “Description” identify and document the class,  
and should not be overridden.

1       “MaximumScripts” specifies the maximum number of scripts that will be run  
2 from any one instance of the consumer before starting a new instance. Default  
3 value: 300. A value of zero or NULL will result in the default being used.

4       “Timeout” specifies the maximum amount of time in minutes that the  
5 consumer will be allowed to run before starting a new instance of the consumer. If  
6 zero, lifetime is controlled by the “MaximumScripts” property. Valid Range: 0-  
7 71,000.

8  
9       The primary purpose of the “MaximumScripts” and “TimeOut” properties  
10 is to ensure that the consumer will eventually shut down, thereby removing any  
11 memory or resource leaks caused by poorly written scripts. Failure to load the  
12 scripting engine or parse and validate the script is considered a failure. Error  
13 return code from the script is likewise considered a failure. The active scripting  
14 event consumer will run in a separate process due to its inherent danger.

#### 15 16 SMTP Event Consumer

17       The SMTP event consumer will send an e-mail message via SMTP each  
18 time an event is delivered to the consumer. An SMTP server must exist on the  
19 network for the SMTP event consumer to work properly. The logical consumer  
20 class for the SMTP event consumer is:

```

class SMTPEventConsumer : __EventConsumer
{
    [key] string Name;
    string SMTPServer;
    string Subject;
    string Message;
    [not_null] string ToLine;
    string CcLine;
    string BccLine;
};

```

where

“SMTPServer” is the name of the SMTP server through which mail will be sent. For example, IP addresses, DNS or NetBIOS names can be used to identify the SMTP server.

“Subject” is the template for the subject of the message.

“Message” is the template for the body of the message.

“ToLine” is the semi-colon-separated list of addresses to send the message to.

“CCLine” is the semi-colon-separated list of addresses to CC.

“BccLine” is the semi-colon-separated list of addresses to BCC.

Failure to send mail (error return code from the service) is considered a failure.

### Paging Event Consumer

The paging event consumer will page an arbitrary phone number with an arbitrary message, using industry-standard TAP protocol. No TAPI provider needs to be installed on the server. The logical consumer class for the paging event consumer is:

```

class TAPIEventConsumer : __EventConsumer
{
    [key] string Name;
    string PhoneNumber;
    string ID;
    string Message;
    string Port;
    uint32 BaudRate;
    string ModemSetupString;
    uint32 AnswerTimeout = 30;
};

```

where

“PhoneNumber” is the number to dial. Any non-numeric symbols in this string are ignored.

“ID” is the paging subscriber ID.

“Message” is the alphanumeric message to be sent,

“Port” is the port to which the modem is connected (e.g. “COM1”).

“BaudRate” is the maximum baud rate to use. If left NULL, the maximum available rate will be used.

“ModemSetupString” should be left NULL except when the TAP server is not compliant with the protocol’s suggested defaults.

“AnswerTimeout” is the number of seconds to wait for the server to pick up the phone. The default value is 30 seconds.

Fig. 5 illustrates the forwarding of events from multiple event sources to a common event target. A pair of event sources 402 and 404 each forward certain events (determined by the appropriate filter properties) to an event target 406. In a particular embodiment, event target 406 is a central event logging computer that logs event data from many different event sources. Although Fig. 5 illustrates two



1 event sources 402 and 404, alternate embodiments may include any number of  
2 event sources coupled to event target 406.

3 Each event source 402 and 404 includes an instance of a forwarding  
4 consumer, an instance of a filter, and an instance of a binding that binds the  
5 forwarding consumer to the filter. Event target 406 includes an instance of a log-  
6 to-file consumer, an instance of a filter, and an instance of a binding that binds the  
7 log-to-file consumer to the filter. Events received by or generated by event source  
8 402 or 404 that meet the filter criteria (as defined by the filter properties) are  
9 forwarded by the forwarding consumer to the event target 406 for logging. Events  
10 received by event target 406 may be processed or forwarded to another event  
11 target (not shown) for processing or further forwarding. Thus, a particular event  
12 may be forwarded through multiple devices until a destination device is reached.

#### 13 14 Forwarding Consumer Provider

15 The Forwarding Consumer Provider provides sinks (e.g., a piece of code  
16 that accepts events) for the Data and Event Logical Forwarding Consumer  
17 instances. It exists as a DLL and is an in-proc COM object.

18 The forwarding consumer provider uses the "WbemMessageSender" COM  
19 objects to send messages. This insulates the forwarding consumer provider from  
20 sending messages via MSMQ, Named-Pipes, etc. See the WBEM Messaging  
21 Layer specification for more details.

22 The "IwbemObjectInternals" interface is used for marshaling objects. The  
23 forwarding consumer will not send class information as part of the message. It  
24 will send a classname, decoration and instance data. When sending multiple  
25

objects, their class and decoration data is packaged once. The

“IwbemObjectInternals” interface is an internal COM interface.

Format:

DWORD dwSig; // FCON in ascii  
char cVersionMajor; // 1  
char cVersionMinor; // 0  
char cType; // 0 – for event, 1 for data  
char cLast; // 0 – for FALSE, 1 – for TRUE  
DWORD dwReserved; // not used  
GUID CorrelationId; // not used for event types  
DWORD dwObjs; // num objs in this message  
String ClassName – null terminated.  
Decoration Part // only one.  
Instance Part // dwObjs instances

Table 2 below is used to determine the target queue when one is not specified.  
See Forwarding Queues for more details.

TABLE 2

Sender Operation Mode	Delivery Class	Target Queue
N/A	Synchronous	N/A
On-Line	Express	Public, Private(D)
On-Line	Guaranteed	Public Guaranteed, Private Guaranteed(D)
Off-Line domain member	Express	Private (D)
Off-Line domain member	Guaranteed	Private Guaranteed(D)
Off-Line workgroup	Express	Private(D)
Off-Line workgroup	Guaranteed	Private Guaranteed(D)

(D) – Direct MSMQ Format Name

## Event Forwarding Consumer

Each event or event batch that is indicated to the consumer will be packaged into a single message and sent to the target.

## Data Forwarding Consumer

When obtaining the results of the query, the results will be packaged into one or more messages. The number of objects in a message depends on their size. The granularity of these messages is around 512K. There will be an indication in the last message that it is the last one. This is set in the "last" property of the forwarded consumer message. This will signal the event provider to signal a null termination event.

## Forwarded Message Provider

There are two types of forwarding queues: Express and Guaranteed. Both queues have identical properties, they are just serviced differently. When the machine is online, there will be two MSMQ queues for each type: Public and Private. When the machine is offline, there will be only one MSMQ queue for each type: Private.

The reason for the two queues in the online case is as follows. If the sender is offline, then it has no way of knowing what queues the receiver has. If we always create/open and service both the public and private queues, then there should not be any problems. The sender would always send to the private queues in this case. There is virtually no extra overhead in servicing the extra queues

1 since overlapped i/o can be used. All of the queues are configured externally  
2 through MSMQ provider or through the MSMQ snap-in.

3 Queue Initialization occurs the first time that the CIMV2 namespace goes  
4 active. See below for more details. If online at the time of initialization, then the  
5 public and private versions of the queue will be created. If offline at the time of  
6 initialization, then only the private versions of the queue will be created.

### 7 8 Forwarding Event Provider

9 Forwarding Event Provider is called regardless of its activation status. Its  
10 "ProvideEvents()" will always be called with a sink. When receiving this sink, the  
11 forwarding event provider starts servicing the queues. Since only guaranteed and  
12 express queues are serviced, there is little overhead in servicing them when no  
13 device is interested in the queues. The queues are serviced using the express and  
14 guaranteed receivers supplied by the "WbemMessageReceiver" layer.

### 15 16 WMI Event Forwarding

17 WMI Event Forwarding refers to the process of subscribing to WMI Events  
18 that are signaled on one machine and directing them to be signaled as WMI Events  
19 on another machine. At a high level, this is accomplished by subscribing a  
20 standard WMI Event Consumer, called the Event Forwarding Consumer, to the  
21 events to be forwarded. The action taken by this consumer when it is notified of  
22 events is to forward the events to remote machines. For each forwarded event that  
23 is received at the destination machine, a new WMI Event, called a Forwarded  
24 Event, is created that contains the original event and is signaled. Consumers

1 interested in events that are forwarded to a machine, subscribe to Forwarded  
2 Events on that machine.

### 3 4 Event Forwarding Consumer

5 An event forwarding consumer is subscribed to events using the normal  
6 WMI event model. An instance of the event forwarding logical consumer is  
7 created and is bound to an event filter that describes the events that are to be  
8 forwarded. The way that the event forwarding consumer instance is configured  
9 directs the system on how to forward events. Destination addresses, Quality of  
10 Service (QoS), and security information are examples of configuration information  
11 exposed to the user.

12 The following is an abstract definition of a forwarding consumer. In later  
13 versions, there may be forwarding consumers that forward messages other than  
14 events. For now we are concerned with forwarding events.

15 [abstract]  
16 class MSFT\_ForwardingConsumer : \_\_EventConsumer  
17 {  
18 [KEY] STRING NAME;  
19 string Targets[];  
20 [values{ 1, 2, 3, 4 },value\_map{ "Synchronous", "Express",  
21 "Guaranteed", "Transactional"}]  
22 sint32 ForwardingQoS = 2;  
23 boolean Authenticate = TRUE;  
24 boolean Encryption = FALSE;  
25 string TargetSD;  
26 };  
27 class MSFT\_EventForwardingConsumer : MSFT\_ForwardingConsumer  
28 {  
29 };  
30

31 where

1       “Name” is the key property identifying the instance of a  
2 “MSFT\_ForwardingConsumer”.

3       “Targets” identifies the destinations of the forwarded messages. This  
4 property is an array, so it can contain multiple destinations.

5       “ForwardingQoS” specifies the QoS to be used for the forwarding.

6       “Authenticate” tells the sender if authentication information needs to be  
7 part of the message. If the sending machine belongs to a workgroup, then this  
8 property is ignored.

9       “Encryption” tells the sender to encrypt the message body before sending.  
10 If the sending machine belongs to a workgroup, then this property is ignored.  
11 Encryption can be used when the sending machine is on-line.

12       “TargetSD” is a textual representation of a security descriptor using SDDL.  
13 This security descriptor is used for controlling which security identities can  
14 subscribe to the forwarded event at the receiving end.

## 15 16 Forwarded Events

17       A forwarding consumer forwards information using a forwarded message.  
18 When a forwarded message is received at the destination it is surfaced using a  
19 WMI Event. This event is defined in the root\cimv2 namespace and its schema  
20 looks like:

```
21       class Win32_WmiForwardedMessageEvent : __ExtrinsicEvent  
22       {  
23           datetime Time;  
24           string Machine;  
25           string Account;  
          boolean Authenticated;  
      };
```

where

“Time” is the time the message was sent.

“Machine” is the machine the message was sent from.

“Account” is the Security Account the message was sent under.

“Authenticated” states whether the message was authenticated by the receiver.

This class is intended to be overridden by concrete event types. Since the concern is with forwarding events, here, a concrete event class is defined which is derived from the “Win32\_WmiForwardedMessageEvent”.

```
class Win32_WmiForwardedEvent : Win32_WmiForwardedEvent
{
    __Event Event;
};
```

where

“Event” is the original event that was delivered to the forwarding consumer on the sender.

Events can be forwarded using a synchronous, express, or guaranteed quality of service (QoS). Synchronous QoS means that the notification of the event at the sender, the forwarding of the event to the destination, and the signaling of the event at the receiving end all take place in the same execution path. RPC communication is used for this purpose. By definition, this type of forwarding is guaranteed. However, RPC is limited in that the sending machine requires network connectivity to the receiver at the time of forwarding. In a

1 particular embodiment, the forwarding consumer uses DCOM for synchronous  
2 communication.

3 The three QoS classes listed below are called asynchronous because the  
4 forwarding of the event is not in the same execution path as the notification of the  
5 event. Unlike synchronous forwarding, asynchronous forwarding uses messaging  
6 communication rather than RPC. This has the advantage of being able to forward  
7 events even when the sender and receiver are disconnected. The following are the  
8 Asynchronous QoS classes:

9  
10 Express. Express forwarding makes no guarantees that the message will be  
11 received at the destination. If an error is encountered in forwarding then the event  
12 can be discarded. Because of the absence of a guarantee, however, express  
13 delivery is faster than any other asynchronous QoS.

14 Guaranteed. The guaranteed forwarding subsumes the express QoS class  
15 and also provides a guarantee that the event will make it to the destination at least  
16 once. This guarantee applies across machine and network failures.

17 Transactional. Transaction forwarding subsumes the guaranteed QoS class  
18 and also provides a guarantee that an event will make it to the destination at most  
19 once.

20  
21 The forwarding consumer uses MSMQ for Messaging Communication.  
22 Using MSMQ allows the forwarding consumer to support offline forwarding and  
23 store-and-forward for all asynchronous QoS classes.

24 Store-and-forward refers to the ability for a message to be forwarded to a  
25 remote destination machine even when the destination machine is unreachable or



1 down. It also allows a message to be forwarded to the destination when the source  
2 machine is unreachable or down. This means that a message can reach its  
3 destination even when the source or destination machines are never reachable or  
4 up at the same time. This is accomplished by forwarding the message to an  
5 intermediate machine if the destination is not reachable. Store-and-forward is  
6 automatically the class of delivery used when forwarding messages to a remote  
7 destination using an asynchronous QoS class.

8 Offline forwarding is the ability for a machine to forward messages without  
9 being connected to the network. The messages are stored locally, but when the  
10 machine goes back online, the messages are automatically forwarded. This is  
11 different than store-and-forward because, in this case, the sender does have  
12 connectivity to any machine, even the intermediate one used for store-and-  
13 forward. The two features can work together though. For example, it is possible  
14 that when the sending machine does come online, the receiving machine is down.  
15 In this case, the store-and-forward feature would be activated when the sending  
16 machine came online.

17 The target property of a forwarding consumer can contain one or more  
18 destination addresses. Each address is represented in one of three formats:  
19 network, indirect, or an MSMQ format name. When a forwarding consumer needs  
20 to send a message and there are multiple destinations specified, then the message  
21 will be forwarded to the targets in their order of appearance until the send is  
22 successful. A successful message send depends on the delivery class.

23 A network target name is any valid IP address, NetBIOS name, or DNS  
24 name. For synchronous forwarding QoS, a network target name is used to  
25 perform communication over DCOM. For asynchronous forwarding QoS, a

1 network target name is used to perform communication over MSMQ. In this case,  
2 the target MSMQ format name will be derived from the network target name and  
3 the delivery class property.

4 For most cases, the user will configure network target names and the  
5 forwarding consumer will derive the low-level address of the target based on how  
6 it is configured. To override this, the user can specify a low-level MSMQ target  
7 name.

8 There are three types of MSMQ format names. These are public, private  
9 and direct.

10 Public - identifies a queue using a queue GUID.

11 Private - identifies a queue using a machine GUID and queue identifier.

12 Direct - identifies a queue using a protocol, queue location, and queue  
13 logical name.

14 Each type of format name has implications for sending messages. Table 3  
15 below describes these implications.

16  
17 TABLE 3

Format / Implications	Requires Sender to be operating in On- Line mode when sending messages	Requires Sender and Receiver to be part of same forest (or MSMQ enterprise for non w2k domains )	Supports Store-And- Forward functionality
Public	Yes	Yes	Yes
Private	No	Yes	Yes
Direct	No	No	No

22 A valid Target MSMQ Format Name is any valid MSMQ format name  
23 prefixed with MSMQ! .  
24  
25

1 An indirect target name can be used for indirect addressing. An indirect  
2 target name is any valid WMI instance object path prefixed by WMI! and suffixed  
3 with !<PropertyName> . When specified, the forwarding consumer will obtain the  
4 object specified in the address and use the specified property to determine the  
5 resolved address. The resolved address can be any valid network or MSMQ  
6 address, or list of valid network or MSMQ addresses. It cannot be another indirect  
7 address.

8 The following is an example of a valid indirect target name:

9  
10 wmi!\\mymachine\root\default:myclass="myinstance":myprop  
11

12 When an indirect target name is encountered, the value of the specified property is  
13 obtained. The type of this property must be a string or an array of strings. In both  
14 cases, the strings are treated as if they were explicitly listed in the targets property.

15 Each machine that can accept forwarded messages will have one or more  
16 well-known entry points. For MSMQ, these entry points are MSMQ queues. For  
17 DCOM, this entry point is a DCOM server object that is implemented by the  
18 forwarding event provider.

19 It is typically not necessary to perform any configuration on the receiver  
20 end of the forwarding consumer. However, in some circumstances, it may be  
21 necessary to query certain properties of the entry points and on even rarer  
22 occasions, be able to modify them. The following are the reasons why a user  
23 might need knowledge of the messaging entry points on the receiver:

24  
25 To set the queue disk quota for MSMQ entry points.

1 To obtain queue address information for manual configuration of the target  
2 address at the sending end.

3  
4 Both of these are specific to MSMQ. The MSMQ provider will model these  
5 queues. The MSMQ snap-in can also be used to access these queues. Both of  
6 these can perform the actions described above.

### 7 8 MSMQ Queues

9 All queues will have a default quota of 10 Meg.

10 All queues will have a security descriptor that allows only LocalSystem and  
11 administrators read and modify access, and allows everyone send access.

12 An administrator can distinguish between MSMQ Queues used for  
13 forwarding and those used by other applications using the queue type property.

14 Queues used for event forwarding will have a queue type of:

15  
16 {BD29DFFF-7553-4d3b-8401-4646AC9A70C6}

17  
18 Each forwarding queue is either public or private. A public queue can be  
19 referenced through a public or direct format name. A public queue can only exist  
20 on machines that are online. A Private Queue can only be referenced through a  
21 private or direct format name. There are no restrictions on the machine that  
22 private queues are created on. Furthermore, there are authenticated and non-  
23 authenticated versions of these queues.

24 When determining endpoint address information, the user must decide  
25 which of these types of queues they will want to use. The public/private and

1 authentication properties of a queue each have their own set of requirements on  
2 how they can be used. This primarily depends on MSMQ installation type.

3 When a forwarding consumer executes, it generates trace events to facilitate  
4 debugging. There are two types of trace events:

5  
6 MSFT\_ForwardingConsumerTraceEvent; and  
7 MSFT\_ForwardingConsumerTargetTraceEvent  
8

9 For each event that is delivered to a forwarding consumer, there will be one  
10 MSFT\_ForwardingConsumerTraceEvent generated which states the outcome of  
11 the execution. For each target tried during this execution, there will be one  
12 MSFT\_ForwardingConsumerTargetTraceEvent generated stating the outcome of  
13 the forwarding to that particular target. The schema for these two classes as well  
14 as their base class is described below:

15  
16 class MSFT\_ForwardingConsumerTraceEventBase : \_\_ExtrinsicEvent  
17 {  
18 MSFT\_ForwardingConsumer Consumer;  
19 \_\_Event Event;  
20 string ExecutionId;  
21 uint32 StatusCode;  
22 };  
23

24 This is the base class that all forwarding consumer trace events derive from.

25 “Consumer” is the forwarding consumer instance.

“Event” is the event that triggered the forwarding consumer.

“ExecutionId” is a GUID that is generated each time a forwarding  
consumer is delivered an event.

1       “StatusCode” contains the outcome of the execution of the forwarding  
2 consumer.

```
3  
4       class           MSFT_ForwardingConsumerTraceEvent               :  
5       MSFT_ForwardingConsumerTraceEventBase  
6       {  
7       string TargetUsed;  
8       boolean Queued;  
9       };
```

10       Each time a forwarding consumer executes, an instance of this event is  
11 signaled.

12       “TargetUsed” contains the address of the target that was used to  
13 successfully forward the message. This property is NULL when “StatusCode”  
14 specifies an error.

15       “Queued” states whether the event was forwarded using RPC or was  
16 queued. This property will be NULL when “StatusCode” specifies an error.

```
17       class  
18       MSFT_ForwardingConsumerTargetTraceEvent:MSFT_ForwardingConsumerTrac  
19       eEventBase  
20       {  
21       string Target;  
22       };
```

23       “Target” specifies the address of the target that was used to attempt to  
24 forward an event. A “ForwardingConsumerTraceEvent” can be correlated with its  
25 “ForwardingConsumerTargetTrace” events using the “ExecutionId” parameter.

When forwarding using a synchronous QoS, all errors that can occur are  
detected at the time of forwarding. With asynchronous forwarding, configuration  
errors can usually be detected at the time of forwarding as well. For these reasons,

1 most errors can be detected by subscribing to the trace events described above  
2 (where status code specifies an error).

3 There are some error cases with asynchronous forwarding that cannot be  
4 detected at the time of forwarding. These errors are usually detected much later.  
5 For this reason, an event is provided that will alert any subscribers that an  
6 asynchronous error has occurred in forwarding. This event is:

```
7 class Win32_WmiForwardedAckEvent : Win32_WmiForwardedMessageEvent
8     {
9         Event Event;
10        uint32 Status;
11        string Target;
12        uint32 QoS;
13        boolean Authentication;
14        boolean Encryption;
15        string ConsumerPath;
16        string ExecutionId;
17    };
18
```

19 Note that it is derived from the forwarded message event so it possesses all  
20 of its properties and semantics as well.

21 “Event” is the original event that was forwarded.

22 “Status” contains the error code for the reason why the forwarded event  
23 was returned.

24 “QoS” contains the value of the QoS parameter when the message was  
25 forwarded.

“Authentication” contains the value of the auth parameter when the  
message was forwarded.

“ConsumerPath” contains the relpath of the forwarding consumer that was  
responsible for forwarding the message.

1       “ExecutionId” contains the ExecutionId that was used to forward the  
2 original event. This id would be the same as the one contained in the  
3 “ForwardingConsumerTraceEvent” that was caused by the original event being  
4 forwarded.

#### 5 6 Forwarding Security

7       For messages that are forwarded synchronously over DCOM, then DCOM  
8 security is used for authentication. For messages that are forwarded  
9 asynchronously over MSMQ, then MSMQ security is used for authentication.  
10 With respect to authentication, there are two types of entry points for receiving  
11 messages: authenticated and unauthenticated. Any message that is accepted by the  
12 authenticated entry point must have authentication information associated with it.  
13 This is a responsibility of the sender. In other words, the sender’s forwarding  
14 consumer must have the authentication property set to “TRUE” in order to send to  
15 an authenticated entry point (queue). For both types of security, the sender and  
16 receiver must be part of the same forest (or MSMQ enterprise in non-w2k  
17 domains) for authentication to be possible.

18       A forwarded event will contain the security identity of the sender, if  
19 available. There is also a Boolean property on the event that specifies if the event  
20 has been authenticated. An unauthenticated forwarded event may still contain the  
21 identity of the sender, but only when the authenticated property is set to “TRUE”  
22 is this property to be believed. Forwarded events that are returned to the sender  
23 asynchronously because of some failure, are always verified that they actually  
24 originated from the sender.



1 WMI events support access control on subscriptions and on events. Access  
2 control on subscriptions state the security identity of the event providers that the  
3 subscriber is willing to receive events from. Access control on events state what  
4 security identities can subscribe to the event. When a forwarded event is signaled  
5 on the receiver, the id of the sender will be used to perform the access check  
6 controlling access to subscribers. If the forwarded event has not been  
7 authenticated, then delivery of this event will only occur to subscribers who allow  
8 "everyone" access. The forwarded event will also be signaled with a security  
9 descriptor that is passed from the sender.

#### 11 Sender

12 For all types of forwarding QoS, if the authenticate property is set to  
13 "TRUE" on the forwarding consumer, then it will attach authentication info with  
14 the message and send it to the authenticated entry point on the target. If the  
15 property is set to "FALSE", then the message is sent to the unauthenticated entry  
16 point on the target.

17 The sending identity of forwarded messages depends on two factors:  
18 platform and type of forwarding consumer. For event forwarding consumers, the  
19 identity of the message depends on the "MaintainSecurityContext" property of the  
20 binding to the forwarding consumer. If the "MaintainSecurityContext" property  
21 is "TRUE", then the security principal attached to the forwarded messages will be  
22 the same as the event providers. If the "MaintainSecurityContext" property is  
23 "FALSE" or in the case of a data forwarding consumer, the message is sent using  
24 the account that winmgmt is running under. For all NT platforms, this account is  
25 Localsystem. On win2k and higher platforms this identity be used for

1 authentication. On older platforms, authentication cannot occur unless the  
2 forwarding consumer is run out of proc to winmgmt. This is possible, but will  
3 require that the administrator adjust the DCOM registration of the forwarding  
4 consumer to run out-of-proc and under a specific domain account.

5 For authenticated asynchronous forwarding, the sending security principal  
6 is registered with MSMQ. When the forwarding consumer is initialized, it will try  
7 to register the account it is running under with MSMQ. Registration occurs when  
8 the machine is running in online mode.

9 Only in the case where "MaintainSecurityContext" is set to "TRUE", the  
10 forwarding consumer is an event Forwarding consumer, the forwarding QoS is  
11 asynchronous, and the platform is not win2k or higher will the administrator be  
12 responsible for registering the account with MSMQ manually.

13 The forwarding consumer will have the ability to specify a DACL to  
14 control what consumers can receive the messages on the receiving end. By  
15 default, there will be no DACL, thereby allowing all consumers to subscribe to  
16 messages that the forwarding consumer sends. An administrator must construct  
17 the DACL to set on the forwarding consumer. This can be done using Win32  
18 Security APIs or through scripting helper objects.

19 A forwarding consumer will encrypt all messages that it sends when its  
20 "Encrypt" property is set to "TRUE". Encryption of messages cannot be  
21 performed when the sending machine is offline. Encryption cannot be used when  
22 forwarding messages to a machine outside the win2k forest (MSMQ enterprise for  
23 non-win2k domains).

24 The forwarding consumer will be part of a WMI installation. The schema  
25 for the forwarding consumer will exist in the root\default namespace. Installation

1 of the schema of the forwarding consumer will be installed by default into the  
2 root\default namespace, but can be installed by the user in other namespaces.

3 The forwarding receiver will support MSMQ independent client and server  
4 installations.

5 There are two modes of operation for the forwarding consumer and receiver  
6 with respect to MSMQ: Online and Offline. These terms refer to connectivity to  
7 an MSMQ Server. There are two types of MSMQ installations: Workgroup and  
8 Domain. A workgroup installation is always treated as Offline.

9 On initialization of winmgmt, domain membership and connectivity to a  
10 DC are checked. If both are true, then winmgmt will operate in Online mode with  
11 respect to MSMQ. If not, then it operates in Offline Mode.

12 Fig. 6 illustrates an example of a suitable operating environment in which  
13 the event management system described herein may be implemented. The  
14 illustrated operating environment is only one example of a suitable operating  
15 environment and is not intended to suggest any limitation as to the scope of use or  
16 functionality of the invention. Other well-known computing systems,  
17 environments, and/or configurations that may be suitable for use with the  
18 invention include, but are not limited to, personal computers, server computers,  
19 hand-held or laptop devices, multiprocessor systems, microprocessor-based  
20 systems, programmable consumer electronics, gaming consoles, cellular  
21 telephones, network PCs, minicomputers, mainframe computers, distributed  
22 computing environments that include any of the above systems or devices, and the  
23 like.

24 Fig. 6 shows a general example of a computer 442 that can be used in  
25 accordance with the invention. Computer 442 is shown as an example of a

1 computer that can perform the various functions described herein. Computer 442  
2 includes one or more processors or processing units 444, a system memory 446,  
3 and a bus 448 that couples various system components including the system  
4 memory 446 to processors 444.

5 The bus 448 represents one or more of any of several types of bus  
6 structures, including a memory bus or memory controller, a peripheral bus, an  
7 accelerated graphics port, and a processor or local bus using any of a variety of  
8 bus architectures. The system memory 446 includes read only memory (ROM)  
9 450 and random access memory (RAM) 452. A basic input/output system (BIOS)  
10 454, containing the basic routines that help to transfer information between  
11 elements within computer 442, such as during start-up, is stored in ROM 450.  
12 Computer 442 further includes a hard disk drive 456 for reading from and writing  
13 to a hard disk, not shown, connected to bus 448 via a hard disk drive interface 457  
14 (e.g., a SCSI, ATA, or other type of interface); a magnetic disk drive 458 for  
15 reading from and writing to a removable magnetic disk 460, connected to bus 448  
16 via a magnetic disk drive interface 461; and an optical disk drive 462 for reading  
17 from and/or writing to a removable optical disk 464 such as a CD ROM, DVD, or  
18 other optical media, connected to bus 448 via an optical drive interface 465. The  
19 drives and their associated computer-readable media provide nonvolatile storage  
20 of computer readable instructions, data structures, program modules and other data  
21 for computer 442. Although the exemplary environment described herein employs  
22 a hard disk, a removable magnetic disk 460 and a removable optical disk 464, it  
23 will be appreciated by those skilled in the art that other types of computer readable  
24 media which can store data that is accessible by a computer, such as magnetic  
25 cassettes, flash memory cards, random access memories (RAMs), read only

1 memories (ROM), and the like, may also be used in the exemplary operating  
2 environment.

3 A number of program modules may be stored on the hard disk, magnetic  
4 disk 460, optical disk 464, ROM 450, or RAM 452, including an operating system  
5 470, one or more application programs 472, other program modules 474, and  
6 program data 476. A user may enter commands and information into computer  
7 442 through input devices such as keyboard 478 and pointing device 480. Other  
8 input devices (not shown) may include a microphone, joystick, game pad, satellite  
9 dish, scanner, or the like. These and other input devices are connected to the  
10 processing unit 444 through an interface 468 that is coupled to the system bus  
11 (e.g., a serial port interface, a parallel port interface, a universal serial bus (USB)  
12 interface, etc.). A monitor 484 or other type of display device is also connected to  
13 the system bus 448 via an interface, such as a video adapter 486. In addition to the  
14 monitor, personal computers typically include other peripheral output devices (not  
15 shown) such as speakers and printers.

16 Computer 442 operates in a networked environment using logical  
17 connections to one or more remote computers, such as a remote computer 488.  
18 The remote computer 488 may be another personal computer, a server, a router, a  
19 network PC, a peer device or other common network node, and typically includes  
20 many or all of the elements described above relative to computer 442, although  
21 only a memory storage device 490 has been illustrated in Fig. 6. The logical  
22 connections depicted in Fig. 6 include a local area network (LAN) 492 and a wide  
23 area network (WAN) 494. Such networking environments are commonplace in  
24 offices, enterprise-wide computer networks, intranets, and the Internet. In certain  
25 embodiments, computer 442 executes an Internet Web browser program (which

1 may optionally be integrated into the operating system 470) such as the "Internet  
2 Explorer" Web browser manufactured and distributed by Microsoft Corporation of  
3 Redmond, Washington.

4 When used in a LAN networking environment, computer 442 is connected  
5 to the local network 492 through a network interface or adapter 496. When used  
6 in a WAN networking environment, computer 442 typically includes a modem 498  
7 or other means for establishing communications over the wide area network 494,  
8 such as the Internet. The modem 498, which may be internal or external, is  
9 connected to the system bus 448 via a serial port interface 468. In a networked  
10 environment, program modules depicted relative to the personal computer 442, or  
11 portions thereof, may be stored in the remote memory storage device. It will be  
12 appreciated that the network connections shown are exemplary and other means of  
13 establishing a communications link between the computers may be used.

14 Computer 442 typically includes at least some form of computer readable  
15 media. Computer readable media can be any available media that can be accessed  
16 by computer 442. By way of example, and not limitation, computer readable  
17 media may comprise computer storage media and communication media.  
18 Computer storage media includes volatile and nonvolatile, removable and non-  
19 removable media implemented in any method or technology for storage of  
20 information such as computer readable instructions, data structures, program  
21 modules or other data. Computer storage media includes, but is not limited to,  
22 RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM,  
23 digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic  
24 tape, magnetic disk storage or other magnetic storage devices, or any other media  
25 which can be used to store the desired information and which can be accessed by

1 computer 442. Communication media typically embodies computer readable  
2 instructions, data structures, program modules or other data in a modulated data  
3 signal such as a carrier wave or other transport mechanism and includes any  
4 information delivery media. The term "modulated data signal" means a signal that  
5 has one or more of its characteristics set or changed in such a manner as to encode  
6 information in the signal. By way of example, and not limitation, communication  
7 media includes wired media such as wired network or direct-wired connection,  
8 and wireless media such as acoustic, RF, infrared and other wireless media.  
9 Combinations of any of the above should also be included within the scope of  
10 computer readable media.

11 The invention has been described in part in the general context of  
12 computer-executable instructions, such as program modules, executed by one or  
13 more computers or other devices. Generally, program modules include routines,  
14 programs, objects, components, data structures, etc. that perform particular tasks  
15 or implement particular abstract data types. Typically the functionality of the  
16 program modules may be combined or distributed as desired in various  
17 embodiments.

18 For purposes of illustration, programs and other executable program  
19 components such as the operating system are illustrated herein as discrete blocks,  
20 although it is recognized that such programs and components reside at various  
21 times in different storage components of the computer, and are executed by the  
22 data processor(s) of the computer.

23 Although the description above uses language that is specific to structural  
24 features and/or methodological acts, it is to be understood that the invention  
25

defined in the appended claims is not limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the invention.